



Product:	Packets and FunctionDefinitions
Product version:	V1.1
Document ID:	UM-Packets and FunctionDefinitions
Doc revision:	A
Written/Appr.:	RE / SL
Date:	16. Oct. 2008

Industrial Control Design AS

Request:

Function code	1 byte	0x17 (23 decimal)
Read Starting address	2 bytes	0x0000 to 0xffff
Quantity to read	2 bytes	0x0001 to approx. 0x0076
Write starting address	2 bytes	0x0000 to 0xffff
Quantity to write	2 bytes	0x0001 to approx. 0x0076
Write byte count	1 byte	2 x N ₁
Write registers value	N ₁ x 2 bytes	

Response:

Function code	1 byte	0x17 (23 decimal)
Byte count	1 byte	2 x N ₁
Read registers value	N ₁ x 2 bytes	

Packets and FunctionDefinitions User Manual

The content of this document is confidential information not to be published without the consent of Industrial Control Design AS.

Industrial Control Design AS, www.icd.no, support@icd.no, forum.icd.no

Contents

1. INTRODUCTION.....	3
1.1. About.....	3
1.2. DefinitionManager.....	3
<hr/>	
2. PACKETS EXPLAINED.....	4
2.1. Packet.....	4
2.1.1. Description.....	4
2.1.2. Example XML.....	4
2.1.3. Elements.....	4
2.2. Inputs.....	5
2.2.1. Description.....	5
2.2.2. Example XML.....	5
2.2.3. Elements.....	5
2.3. Outputs.....	5
2.3.1. Description.....	5
2.3.2. Example XML.....	6
2.3.3. Elements.....	6
<hr/>	
3. FUNCTIONDEFINITIONS EXPLAINED.....	7
3.1. FunctionDefinition.....	8
3.1.1. Description.....	8
3.1.2. Example XML.....	8
3.1.3. Elements.....	8
<hr/>	
4. CONFIGURATION.....	11

1. Introduction

1.1. About

This document describes how to set up and use Packets and FunctionDefinitions in CDP. Function definitions are used by some CDP I/O-Servers to define how to wrap signals before sending them and how to unwrap signals after receiving them. This enables an IOserver to be flexible and configurable, and will to some extent be compatible with future extensions to a protocol.

1.2. DefinitionManager

Serial, Siemes S7 and Modbus IO-servers use Function definitions, as loaded by the DefinitionManager in Application.xml:

```
<Definitions>  
  <Definition src="Models\ModbusFunctions.xml"></Definition>  
</Definitions>
```

The above lines are a directive to the DefinitionManager that it should load all the definitions it finds in the file ModbusFunctions.xml found in the models folder. Whenever an I/O Server requires a FunctionDefinition, the DefinitionManager will look it up in its loaded definitions to see if it matches. If it does not find a match, an error is printed.

2. Packets explained

When data is to be transported with a request-reply structure, such as the MODBUS protocol¹, the packet structure is an ideal way to set up which data to send and receive.

A Packet is a convenient way for CDP to group signals in an IOserver. A packet wraps channels/signals (with name, type and scaling). It also describes how the data (signals) are to be sent over a transport medium through the use of FunctionDefinitions (explained in the next chapter).

A packet is defined in XML like this:

```
<Packet Name="AnalogIOSignals" FunctionCode="MODBUSReadWriteMultipleRegisters" NetworkConvert="1">
  <Inputs>
    <ChannelGroup Type="Analog" NumberOf="2" Offset="192" ModuleNr="0">
      <Channel Nr="0" Type="short" Name="WinchMotorCtrl_I">
      </Channel>
    </ChannelGroup>
  </Inputs>
  <Outputs>
    <ChannelGroup Type="Analog" NumberOf="2" Offset="576" ModuleNr="0">
      <Channel Nr="0" Type="unsigned short" Name="AI0_0_Cfg">
      </Channel>
    </ChannelGroup>
  </Outputs>
</Packet>
```

2.1. Packet

2.1.1. Description

Contains the Inputs to receive and the outputs to send.

2.1.2. Example XML

```
<Packet Name="IOSignals" FunctionCode="MODBUSReadWriteMultipleRegisters" NetworkConvert="1">
  <Inputs>
    <ChannelGroup Type="Analog" NumberOf="2" Offset="0" ModuleNr="3">
      <Channel Nr="0" Type="short" Name="750-454 AI1" Min="0" Max="32767"></Channel>
      <Channel Nr="1" Type="short" Name="750-454 AI2" Min="0" Max="32767"></Channel>
    </ChannelGroup>
    <ChannelGroup Type="Digital" NumberOf="2" Offset="2" ModuleNr="1" SizeInBytes="2">
      <Channel Nr="0" Type="bool" Name="750-403 DI1"></Channel>
      <Channel Nr="1" Type="bool" Name="750-403 DI2"></Channel>
    </ChannelGroup>
  </Inputs>
  <Outputs>
    <ChannelGroup Type="Analog" NumberOf="2" Offset="0" ModuleNr="0">
      <Channel Nr="0" Type="short" Name="750-554 AO1" Min="0" Max="32767"></Channel>
      <Channel Nr="1" Type="short" Name="750-554 AO2" Min="0" Max="32767"></Channel>
    </ChannelGroup>
    <ChannelGroup Type="Digital" NumberOf="2" Offset="2" ModuleNr="2" SizeInBytes="2">
      <Channel Nr="0" Type="bool" Name="750-501 DO1"></Channel>
      <Channel Nr="1" Type="bool" Name="750-501 DO2"></Channel>
    </ChannelGroup>
  </Outputs>
</Packet>
```

2.1.3. Elements

Packet Attributes	Description
Name	Application-wide unique name for the packet
FunctionCode	A function code to look up in the DefinitionManager. The FunctionCode defines how the input and output data are put into the data-packet that is sent to the device.

¹See *UM-Modbus Setup.pdf*

Packet Attributes	Description
NetworkConvert	Set to 1 to do byte conversion (network conversion) on data bigger than one byte.

2.2. Inputs

2.2.1. Description

Contains the ChannelGroups and Channels / signals to receive input from the physical input from the device we communicate with.

2.2.2. Example XML

```
<Inputs>
  <ChannelGroup Type="Analog" NumberOf="2" Offset="0" ModuleNr="0">
    <Channel Nr="0" Type="short" Name="Input module 0.0"></Channel>
    <Channel Nr="1" Type="short" Name="Input module 0.1"></Channel>
  </ChannelGroup>
</Inputs>
```

2.2.3. Elements

Element	Description
ChannelGroup	An enclosing element for a group of channels/signals.
Channel	A Signal / Channel that correspond to one value received in.

ChannelGroup	Description
Type	Can be 'Analog' or 'Digital'
NumberOf	The Number of channels in this ChannelGroup
ModuleNr	The physical module number, used only for documentation, such as in the WebInterface.
Offset	The ChannelGroup Offset absolute value. Note that Offsets must be continuous, no 'gaps' are allowed. The Offset is used for instance in Modbus communication to indicate register(start) of a read or write operation.
SizeInBytes	Only for Digital ChannelGroups: Specifies the size in bytes of the ChannelGroup. This is usually set to 2 , but please note that for digital packets sent with the MODBUSReadCoils /MODBUSWriteCoils FunctionCodes, you must set SizeInBytes to the actual size in bytes that the digital I/O's use (i.e. <=8 bits requires 1 byte, >8<=16 bits requires 2 bytes etc.).

Channel Attribute	Description
Number	The number in a sequence beginning at 0, must be last channel number+1. On Digital channels this also signifies the bit position.
Type	The c++ data type, can be bool, char, byte, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, float or double. Typical values are bool and short.
Name	The signal name for this channel. Feel free to use more understandable names than what is used in the example, like 'Pressure', 'Oil Level' and so on.

2.3. Outputs

2.3.1. Description

Contains the ChannelGroups and Channels/signals to send as output from the physical modules.

2.3.2. Example XML

```

<Outputs>
  <ChannelGroup Type="Analog" NumberOf="2" Offset="0" ModuleNr="0">
    <Channel Nr="0" Type="short" Name="module out 0.0"></Channel>
    <Channel Nr="1" Type="short" Name="module out 0.1"></Channel>
  </ChannelGroup>
</Outputs>

```

2.3.3. Elements

Element	Description
ChannelGroup	An enclosing element for a group of channels/signals.
Channel	A Signal / Channel that correspond to one value received in.

ChannelGroup	Description
Type	Can be 'Analog' or 'Digital'
NumberOf	The Number of channels in this ChannelGroup
ModuleNr	The physical module number
Offset	The ChannelGroup Offset absolute value. Note that Offsets must be continuous, no 'gaps' are allowed. The Offset is used for instance in Modbus communication to indicate register(start) of a read or write operation.
SizeInBytes	Only for Digital ChannelGroups: Specifies the size in bytes of the ChannelGroup. This is usually set to 2, but please note that for digital packets sent with the MODBUSReadCoils /MODBUSWriteCoils FunctionCodes, you must set SizeInBytes to the actual size in bytes that the digital I/O's use (i.e. <=8 bits requires 1 byte, >8<=16 bits requires 2 bytes etc.).

Channel Attribute	Description
Number	The number in a sequence, starting at 0, must be last number+1. On Digital channels this also signifies the bit position.
Type	The c++ data type, can be bool, char, byte, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, float or double. Typical values are bool and short.
Name	The signal name for this channel.

3. FunctionDefinitions explained

A FunctionDefinition can be seen as a wrapper for how the data to send and/or receive is organized. Special data can be put in front of, and/or behind the actual data to send, so that a header and/or footer will be automatically embedded and transmitted with the packet-data. The reason for using a FunctionDefinition is to separate the transport-method/encapsulation from the actual data to send / receive.

An example of modbus function 23 (0x17), “Read Write multiple registers” is set up in the definition below. If you look at page 41 in the “Modbus Application Protocol Specification V1.1” from <http://www.modbus.org/>, you can see that the protocol is organized like this:

Request:

Function code	1 byte	0x17 (23 decimal)
Read Starting address	2 bytes	0x0000 to 0xffff
Quantity to read	2 bytes	0x0001 to approx. 0x0076
Write starting address	2 bytes	0x0000 to 0xffff
Quantity to write	2 bytes	0x0001 to approx. 0x0076
Write byte count	1 byte	2 x N ²
Write registers value	N ² x 2 bytes	

Response:

Function code	1 byte	0x17 (23 decimal)
Byte count	1 byte	2 x N ²
Read registers value	N ² x 2 bytes	

The XML for setting up a FunctionDefinition like this is shown below:

```
<Definition>
  <FunctionDefinition Name="MODBUSReadWriteMultipleRegisters" PacketInput="1" PacketOutput="1">
    <Request>
      <Data Type="byte" Content="23"></Data>
      <Data Type="unsigned short" Content="Input.ChannelGroup0.Offset"></Data>
      <Data Type="unsigned short" Content="Input.NumberOf"></Data>
      <Data Type="unsigned short" Content="Output.ChannelGroup0.Offset"></Data>
      <Data Type="unsigned short" Content="Output.NumberOf"></Data>
      <Data Type="byte" Content="Output.ByteSize"></Data>
      <Data Type="Memory" Content="Output"></Data>
    </Request>
    <Response>
      <Data Type="byte" Content="23"></Data>
      <Data Type="byte" Content="Input.ByteSize"></Data>
      <Data Type="Memory" Content="Input"></Data>
    </Response>
  </FunctionDefinition>
  <!-- More functions can be added below... -->
</Definition>
```

²N = Quantity to Read or Write

3.1. FunctionDefinition

3.1.1. Description

Contains the Request to send and the expected response to receive.

3.1.2. Example XML

```
<FunctionDefinition Name="MODBUSReadWriteMultipleRegisters" PacketInput="1" PacketOutput="1">
  <Request>
    <Data Type="byte" Content="23"></Data>
    <Data Type="unsigned short" Content="Input.ChannelGroup0.Offset"></Data>
    <Data Type="unsigned short" Content="Input.NumberOf"></Data>
    <Data Type="unsigned short" Content="Output.ChannelGroup0.Offset"></Data>
    <Data Type="unsigned short" Content="Output.NumberOf"></Data>
    <Data Type="byte" Content="Output.ByteSize"></Data>
    <Data Type="Memory" Content="Output"></Data>
  </Request>
  <Response>
    <Data Type="byte" Content="23"></Data>
    <Data Type="byte" Content="Input.ByteSize"></Data>
    <Data Type="Memory" Content="Input"></Data>
  </Response>
</FunctionDefinition>
```

3.1.3. Elements

Element	Description
FunctionDefinition	An enclosing element for a Request and a Response
Request	This is the data that is sent across the transport medium. Inside the <Request> tag you specify one or more <Data> tags to indicate how the packet is structured.
Response	This is the data that you expect to receive from the transport medium. Inside the <Response> tag you can specify one or more <Data> tags to indicate how the packet is structured.

FunctionDefinition Attributes	Description
Name	The Application-wide unique name for the definition
PacketInput	Set to 1 if the packet using this definition must have an Inputs element
PacketOutput	Set to 1 if the packet using this definition must have an Outputs element

Request Element	Description
Data	Enclosing element for a data description. The type specifies the size to allocate for the Content tag. The Content tag specifies the data to set.

Data Attributes	Description
Type	The type specifies the size to allocate for the Content. The c++ data types can be bool, char, byte, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, float or double. To specify a block of data taken from the Input or Output, specify 'Memory'
Content	The content to put into this data. See Content Type table below.

Response Element	Description
Data	Enclosing element for a data description. The type specifies the size to allocate for the Content tag. The Content tag specifies the data to set.

Data Attributes	Description
Type	The type specifies the size to allocate for the Content. The c++ data types can be bool, char, byte, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, float or double. To specify a block of data taken from the Input or Output, specify 'Memory'
Content	The content to put into this data. See Content Type table below.

Content Type	Description
byte	8 bits of unsigned data
unsigned char	8 bits of unsigned data - the same as 'byte'
char	8 bits of signed data (7 bits+sign)
short	16 bits of signed data (15 bit+sign / 2 bytes)
unsigned short	16 bits of unsigned data (2 bytes)
long	32 bits of signed data (31 bit+sign / 4 bytes)
int	32 bits of signed data (31 bit+sign / 4 bytes) – the same as 'long'
unsigned long	32 bits of unsigned data (4 bytes)
unsigned int	32 bits of unsigned data (4 bytes) – the same as 'unsigned long'
float	32 bits of floating point data (4 bytes)
double	64 bits of floating point data (8 bytes)
memory	Depends on what is specified in the “Content” tag

Example:

```
<Data Type="byte" Content="23"></Data>
```

This data element will allocate a byte (8 bits) with the (decimal) value of 23 at the position where the <Data> tag is specified.

As can be seen from the Content Type table above, different types have different sizes. The special type “memory” looks in the “Content” tag to determine the size and content of the tag. The “memory” type data can be either “Input” or “Output”. This means that either the “Input” or “Output” section of a packet is put at this position in the actual packet transmitted or received.

For an Input or an Output, you can use the following keywords:

Input/Output keywords	Description
ChannelGroupN	N is a number starting at 0, for instance 'ChannelGroup0'. See table below for additional keywords to use on a ChannelGroup.
ByteSize	The size in bytes that the Input or Output of a packet requires in memory space.
NumberOf	The total register size (ByteSize*2) of an Input or Output
NumberofDigital	Number of Digital channles. Some modbus functions that deal with digital data will need this number instead of 'NumberOf'.

For a given ChannelGroup, the following keywords can be used:

ChannelGroup keywords	Description
Offset	The content of the Offset="" attribute in the Channelgroup
ModuleNr	The content of the ModuleNr="" attribute in the ChannelGroup
NumberOf	The content of the NumberOf="" attribute in the ChannelGroup. Note that NumberOf is actually NumberOfRegisters, so for a Digital Channel with 12 bits (or channels), the NumberOf is actually 1 since a register holds 16 bit. Use NumberofDigital if you want number of Digital channels.
NumberOfDigital	The Number of Digital channels in the Group.
ByteSize	The number of bytes that this ChannelGroup occupies in memory

From the FunctionDefinition that we started with:

```
<Response>  
<Data Type="byte" Content="23"></Data>  
<Data Type="byte" Content="Input.ByteSize"></Data>  
<Data Type="Memory" Content="Input"></Data>  
</Response>
```

The response is expected to be one byte with a constant value of “23”, followed by one byte that is equal to the packet's Input-section ByteSize, and finally that is followed by the actual packet input data (the memory-area for the packet's inputs).

4. Configuration

Setting up FunctionDefinitions is done in XML, and requires some knowledge of the physical data to send and receive to/from a device. The file ModbusFunctions.xml located in the Model folder shows a few examples for setting up FunctionDefinitions.

If you need a special functioncode for your modbus device, it can be set up using a FunctionDefinition.

Example:

Suppose a theoretical Modbus device has a vendor-specific function code 0x7e to set a watchdog timeout. The request is to be sent to register 0x8000, and looks like this:

Request:

Function code	1 byte	0x7e (126 decimal)
Write starting address	2 bytes	0x0000 to 0xffff
Quantity to write	2 bytes	0x0001 to approx. 0x0076
Write byte count	1 byte	2 x N ³
Write registers value	N ³ x 2 bytes	The watchdog timeout in milliseconds.

Response:

Function code	1 byte	0x7e (126 decimal)
Byte count	1 byte	2 x N ³
Read registers value	N ³ x 2 bytes	

The FunctionDefinition for this Modbus PDU will then be:

```
<FunctionDefinition Name="MODBUSSetWatchdogTimeout" PacketInput="0" PacketOutput="1">
  <Request>
    <Data Type="byte" Content="126"></Data>
    <Data Type="unsigned short" Content="Output.ChannelGroup0.Offset"></Data>
    <Data Type="unsigned short" Content="Output.NumberOf"></Data>
    <Data Type="byte" Content="Output.ByteSize"></Data>
    <Data Type="Memory" Content="Output"></Data>
  </Request>
  <Response>
    <Data Type="byte" Content="126"></Data>
    <Data Type="byte" Content="Output.Offset"></Data>
    <Data Type="byte" Content="Output.NumberOf"></Data>
  </Response>
</FunctionDefinition>
```

The packet to send this data correctly should be like this:

```
<Packet Name="WDTimeout" FunctionCode="MODBUSSetWatchdogTimeout" NetworkConvert="1">
  <Outputs>
    <ChannelGroup Type="Analog" NumberOf="1" Offset="0x8000" ModuleNr="0">
      <Channel Nr="0" Type="short" Name="WDTimeout" Value="15"></Channel>
    </ChannelGroup>
  </Outputs>
</Packet>
```

The ChannelGroup has an offset of 0x8000 and one channel containing the Value to write.

³N = Quantity to Read or Write